AJAX IN ACTION

Syntax-Highlights von XSLT 2.0 und XPath 2.0

Dr. Thomas Meinike
Hochschule Merseburg (FH)

07.11.2007 – Frankfurt-Mörfelden

Grundlagen XSLT und XPath

Grundlagen XSLT und XPath

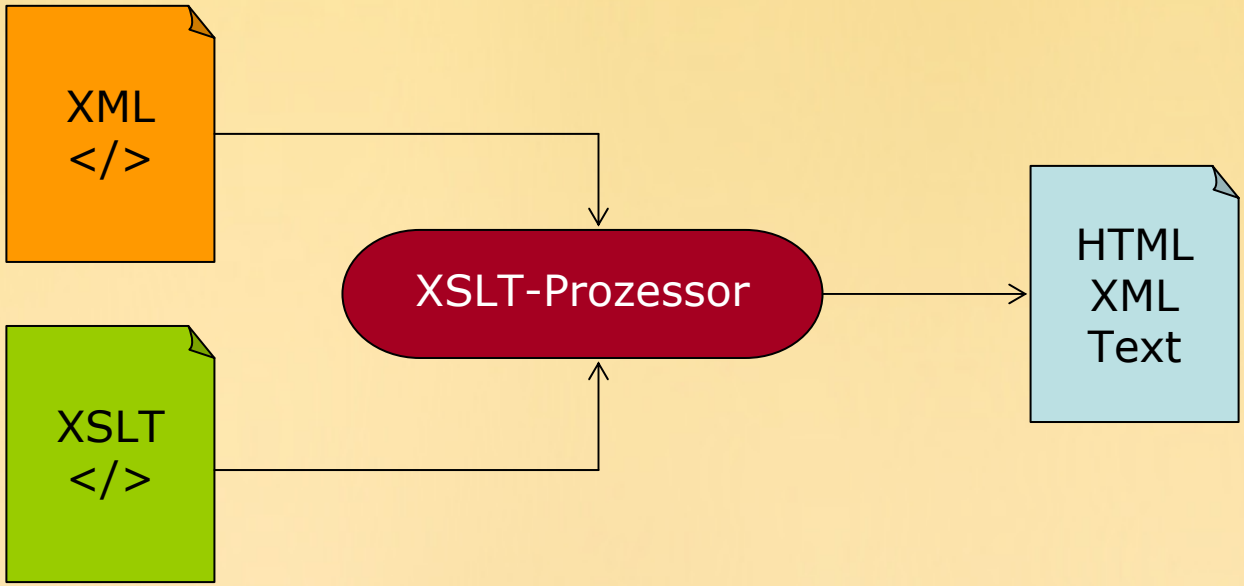
- **XSLT** = Extensible Stylesheet Language (for) Transformations
- W3C-Spezifikationen:
  Version 1.0 → Nov.1999 / 2.0 → Jan. 2007
- Zweck: Verarbeitung von XML-Dokumenten in Zielformate auf Basis von HTML, XML, Text
- gehört neben XSL-FO zur XSL-Familie
- mittlerweile gut etabliert, speziell im Bereich Technische Dokumentation (z.B. Publikation von Print- und Onlineformaten mit DocBook-XML)

Grundlagen XSLT und XPath

- **XSLT** = deklarativ-funktionale Programmiersprache
- jedes Template bildet eine abgeschlossene Einheit und erzeugt zu einer Eingabe aus dem XML-Quellbaum eine Ausgabe im Ergebnisbaum
- keine Seiteneffekte → komplexere Algorithmen (bisher) nur durch rekursive Template-Aufrufe umsetzbar (mit Parameterübergabe)
- benötigte Software: XSLT-Prozessoren auf Client- oder Serverseite (AltovaXML, MSXML, libxslt, Sablotron, Saxon, Xalan, ...)

Grundlagen XSLT und XPath

- **XSLT** – Prinzip der Verarbeitung



Grundlagen XSLT und XPath

- Grundgerüst eines XSLT 1.0-Stylesheets
  (Beispiel für HTML-Ausgabe)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" version="..." encoding="..." .../>

  <xsl:template match="/">

    <html><head><title>...</title></head><body>
      <!-- weitere Template-Aufrufe -->
      <xsl:apply-templates/>
    </body></html>

  </xsl:template> <!-- weitere Templates ... -->

</xsl:stylesheet>
```

Grundlagen XSLT und XPath

- **XPath** = XML Path Language
- W3C-Spezifikationen:
  Version 1.0 → Nov.1999 / 2.0 → Jan. 2007
- Zweck: Lokalisierung von Knoten und Inhalten innerhalb von XML-Strukturen
- XSLT-Abfragen (match, select) verwenden XPath-Ausdrücke (Lokalisierungspfade .../.../..., die aus Lokalisierungsschritten bestehen)
- Version 2.0 bildet eine Untermenge von XQuery 1.0 (W3C, Jan. 2007)

Grundlagen XSLT und XPath

- Aufbau eines Lokalisierungsschrittes
  Achse::Knotentest[Prädikat]

- 13 Achsen beschreiben die Richtung im Dokument (vorwärts / rückwärts), explizite Angabe bei Bedarf

VORWÄRTS	RÜCKWÄRTS	ALLGEMEIN
child	ancestor	attribute (kurz: @)
descendant	ancestor-or-self	namespace
descendant-or-self (kurz: //)	parent (kurz: ..)	
following	preceding	
following-sibling	preceding-sibling	
self (kurz: .)		

- child::x entspricht x a/child::b entspricht a/b
  a/attribute::c entspricht a/@c

Grundlagen XSLT und XPath

- Prädikate ermöglichen Detailabfragen

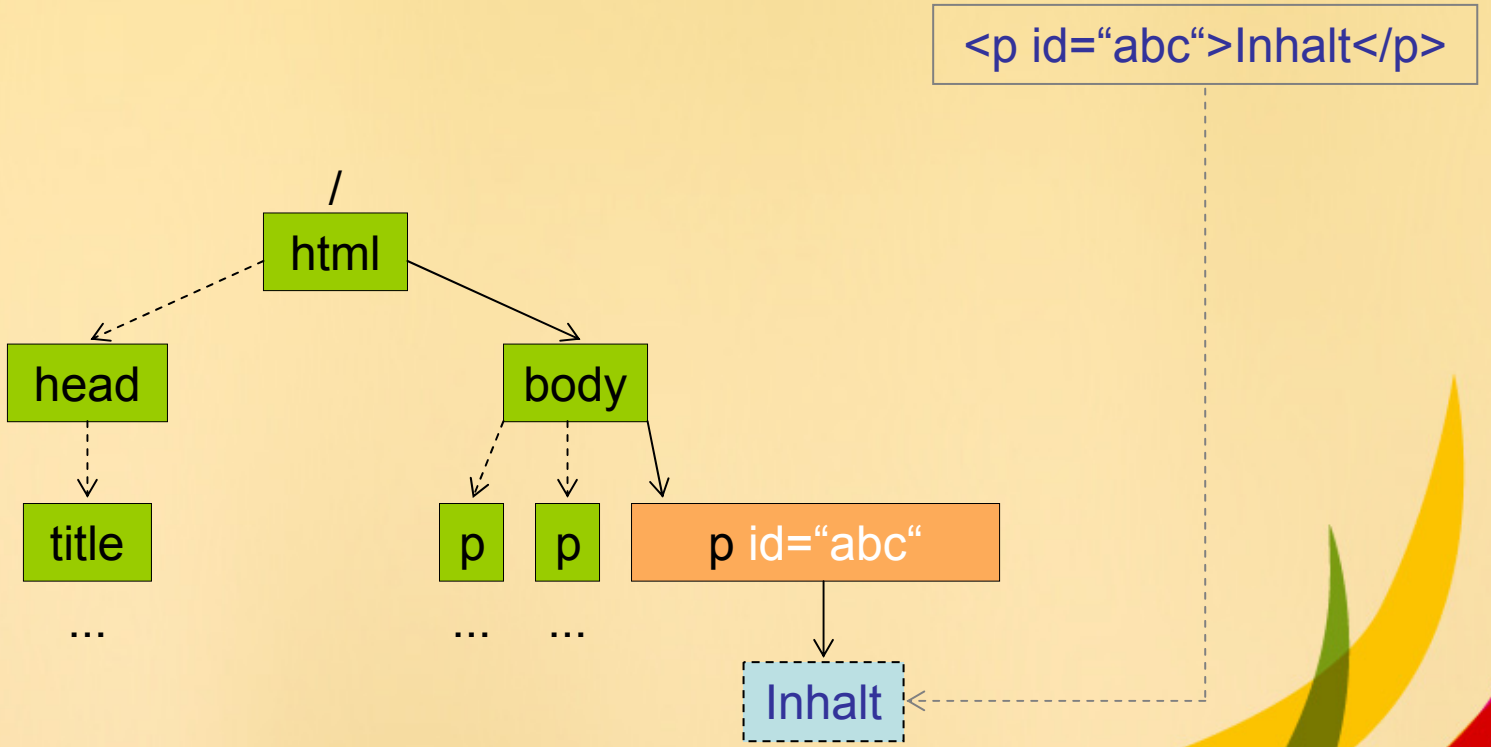
x[y]	Element *x* mit Kindelement *y*
x[a][b] bzw. x[a and b]	Element *x* mit min. einem a- und b-Kindknoten
x[@z]	Element *x* mit Attribut *z*
x[@z='abc']	Element *x* mit Attribut *z* und Wert *abc*
x[@z < 123]	Element *x* mit Attribut *z* und Wert *kleiner 123*
x[3]	drittes Vorkommen in einer *x*-Knotenmenge entspricht x[position()=3]
x[2]/y[1]	erstes y-Element unterhalb des zweiten x-Elem.
x[position()=last()]	letztes Element einer *x*-Knotenmenge
x[substring(.,1,3)='xyz']	Teilzeichenkette des Kontextknotens ist *xyz*

last(), position() und substring() gehören zu
den vordefinierten XPath-Funktionen ab 1.0

Grundlagen XSLT und XPath

- Beispielabfrage in einem XHTML-Dokument
  <xsl:value-of select="/html/body/p[@id='abc']/text()"/>

  → gibt den (Text-)Inhalt des Absatzes mit der ID-Referenz *abc* zurück



Grundlagen XSLT und XPath

- Sieben Knotentypen
  1) Wurzelknoten (/)
  2) Attributknoten
  3) Elementknoten
  4) Kommentarknoten <!-- ... -->
  5) Namensraumknoten
  6) Textknoten
  7) Verarbeitungsanweisungen (PI) <? ... ?>

- Gezielte Abfragen mittels
  comment(), id(), namespace(), node(), text(), processing-instruction()

Grundlagen XSLT und XPath

- XPath 1.0-Operatoren
  - Arithmetik
    + – * div mod
  - Logik
    and or not()
  - Vergleiche (kleiner / größer / gleich / ungleich)
    < <=
    > bzw. > >= bzw. >=
    = !=

Neuerungen in XSLT 2.0 und XPath 2.0

XSLT 2.0 und XPath 2.0

- Relevante Hauptspezifikationen
  - XSL Transformations (XSLT) Version 2.0
    http://www.w3.org/TR/xslt20/
  - XML Path Language (XPath) 2.0
    http://www.w3.org/TR/xpath20/
  - XQuery 1.0 and XPath 2.0 Functions and Operators
    http://www.w3.org/TR/xpath-functions/
  - XQuery 1.0 and XPath 2.0 Data Model (XDM)
    http://www.w3.org/TR/xpath-datamodel/

XSLT 2.0 und XPath 2.0

- Zahlen und Fakten
  - 49 (bisher 35) XSLT-Elemente
  - 19 (bisher 9) XSLT-Funktionen
  - 111 (bisher 27) XPath-Funktionen

    je nach Zählweise werden hier auch Konstruktorfunktionen wie xs:date(…), xs:integer(…) usw. einbezogen

XSLT 2.0 und XPath 2.0

- XSLT 2.0-Grundgerüst

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="fn xs">

<xsl:output method="xhtml" version="..." encoding="..." .../>

  <xsl:template match="/">

    <html><head><title>...</title></head><body>
      <!-- weitere Template-Aufrufe -->
      <xsl:apply-templates/>
    </body></html>

  </xsl:template> <!-- weitere Templates ... -->

</xsl:stylesheet>
```

version="2.0"
+ Namensräume
fn und xs

neue Ausgabe-
methode xhtml
(neben html, xml
und text)

XSLT 2.0 und XPath 2.0

- Sequenzen (überall) ...
  - sind geordnete Listen zur Aufnahme aller möglichen Knotentypen sowie atomarer Werte (untypisiert oder aus dem Bereich der Schema-Datentypen)
  - alle aus dem XML-Baum eingelesenen Strukturen bzw. Inhalte werden in Sequenzen umgewandelt
  - können mit dem (...)-Operator auch direkt erzeugt und mit Werten belegt werden und lassen sich anschließend innerhalb von XPath-Ausdrücken verwenden
  - ersetzen die bisherigen Node-Sets und Ergebnisbaumfragmente (RTF), d. h. Zugriff auf XSLT-Teilbäume (in Variablen) möglich

XSLT 2.0 und XPath 2.0

- Sequenzen direkt erzeugen

```
<xsl:value-of select="(1,2,3,4,5)"/>
<!-- Ergebnis: 1 2 3 4 5 -->

<xsl:value-of select="(1,2,(3,4),5)"/>
<!-- Ergebnis: 1 2 3 4 5 -->

<xsl:value-of select="(1,2,3,4,5)" separator=","/>
<!-- Ergebnis: 1,2,3,4,5 -->

<xsl:value-of select="1 to 5" separator=","/>
<!-- Ergebnis: 1,2,3,4,5 -->

<xsl:value-of select="for $i in (1 to 10)
return $i[. mod 2 = 0]"/>
<!-- Ergebnis: 2 4 6 8 10 -->

<xsl:variable name="seq"
 select="(1, 'abc', 20, xs:date('2007-11-07'))"/>
<xsl:value-of select="$seq[2]"/><!-- Ergebnis: abc -->
```

– neues Attribut separator

– select darf ab 2.0 entfallen, dann Inhalte innerhalb von <xsl:value-of> ... </xsl:value-of> angeben

– Zugriff auf die Sequenz-Inhalte über deren Positionsnummer

XSLT 2.0 und XPath 2.0

- Sequenzen be- und verarbeiten (XPath-Funktionen)
  fn:distinct-values(), empty(), exists(), index-of(), insert-before(), remove(), reverse() und subsequence()

```
<xsl:variable name="seq" select="(1,2,3,4,5,4,3,2,1)"/>

<xsl:value-of select="fn:distinct-values($seq)"/><!-- 1 2 3 4 5 -->

<xsl:value-of select="fn:empty($seq)"/><!-- false -->

<xsl:value-of select="fn:exists($seq)"/><!-- true -->

<xsl:value-of select="fn:index-of($seq,2)"/><!-- 2 8 -->

<xsl:value-of select="fn:insert-before($seq,5,9)"/>
<!-- 1 2 3 4 9 5 4 3 2 1 -->

<xsl:value-of select="fn:remove($seq,1)"/><!-- 2 3 4 5 4 3 2 1 -->

<xsl:value-of select="fn:reverse($seq)"/><!-- 1 2 3 4 5 4 3 2 1 -->

<xsl:value-of select="fn:subsequence($seq,5,3)"/><!-- 5 4 3 -->
```

XSLT 2.0 und XPath 2.0

- Sequenzen aus dem XML-Dokumentbaum

```xml
<?xml version="1.0"?>
<daten>
  <d>10</d>
  <d>20</d>
  <d>30</d>
</daten>
```

```xml
<xsl:template match="daten">

  <xsl:value-of select="d"/>
  <!-- 10 20 30 -->

</xsl:template>
```

Hinweis:
Unter XSLT 1.0 wird nur der erste gefundene Knoten ausgegeben, unter XSLT 2.0 die gesamte d-Sequenz!

XSLT 2.0 und XPath 2.0

- Sequenzen mit xsl:sequence erzeugen

```
<xsl:variable name="seq" as="xs:integer*">
  <xsl:sequence select="(10,20,30,40,50)"/>
  <xsl:sequence select="(60,70,80,90,100)"/>
</xsl:variable>

<xsl:value-of select="$seq"/><!-- 10 20 30 40 50 60 70 80 90 100 -->
```

- Anwendung von Zähl- bzw. Summenfunktion

```
<xsl:value-of select="fn:count($seq)"/><!-- 10 -->

<xsl:value-of select="fn:sum($seq)"/><!-- 550 -->
```

XSLT 2.0 und XPath 2.0

- Unterstützung von XML-Schema-Datentypen
  - 19 Basis-Typen + 5 neu definierte Typen xs:...

anyURI
base64Binary
boolean
date
dateTime
decimal
(mit integer-Subtypen)
double
duration
float
gDay
gMonth
gMonthDay
gYear
gYearMonth
hexBinary
NOTATION
QName
string
time
untyped
untypedAtomic
anyAtomicType
dayTimeDuration
yearMonthDuration

XSLT 2.0 und XPath 2.0

- Unterstützung von XML-Schema-Datentypen
  - Verwendung zur Deklaration und Prüfung von Inhalten

```
<xsl:value-of select="123" as="xs:integer"/>

<xsl:param name="text" select="'Hallo Welt!'" as="xs:string"/>

<xsl:variable name="wert" select="xs:date('2007-11-07')"/>

<xsl:value-of select="$wert instance of xs:date"/><!-- true -->
<xsl:value-of select="$wert instance of xs:double"/><!-- false -->
```

Neues **as**-Attribut für Elemente wie xsl:value-of, xsl:param, und xsl:variable sowie alternative Verwendung von **Konstruktor-funktionen** wie xs:date(...), xs:double, xs:integer(), xs:string(...) usw.

Prüfoperatoren: instance of / cast as / castable as / treat as [...]

XSLT 2.0 und XPath 2.0

- Unterstützung von XML-Schema-Datentypen
  - Basistypen werden generell unterstützt, erweiterte Techniken zur Validierung setzen „schema-aware“-Prozessoren voraus!
    - Saxon 8.9SA (kommerziell – vollständige Schema-Unterstützung)
    - Saxon 8.9B (frei verfügbar – nur Basistypen, keine Schema-Validierung über xsl:import-schema)
    - AltovaXML 2007/2008 (frei verfügbar – vollständige Schema-Unterstützung)

Saxon → Java oder .NET, AltovaXML → native Windows-Anwendung

XPath 2.0-Erweiterungen

XPath 2.0-Erweiterungen

- Neue Operatoren für atomare Wertvergleiche

  eq, ge, gt, le, lt, ne [e: equal, g: greater, l: less, n: not, t: than]

- Neue Operatoren auf Knotenebene

  x << y linker Knoten vor rechtem Knoten
  x >> y linker Knoten nach rechtem Knoten
  x is y Identität von x- und y-Knoten

  fn:deep-equal($seq1,$seq2) prüft auch auf Kindknoten

- Ganzzahlige Division (idiv)

```
<xsl:value-of select="10 div 4"/><!--  2.5 -->
<xsl:value-of select="10 idiv 4"/><!-- 2 -->
```

XPath 2.0-Erweiterungen

- to-Operator zur Sequenz-Konstruktion (s. Folie 18)

```
<xsl:value-of select="1 to 5"/><!-- 1 2 3 4 5 -->
```

- Anwendung mit xsl:for-each spart Rekursionen

  Beispiel zur mehrfachen Ausgabe von Stern-Grafiken

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<hotels>
  <hotel sterne="3">
    <ort>A</ort>
    <name>B</name>
  </hotel>
  <hotel sterne="2">
    <ort>C</ort>
    <name>D</name>
  </hotel>
  <hotel sterne="5">
    <ort>E</ort>
    <name>F</name>
  </hotel>
</hotels>
```

```
<xsl:template match="hotels">

<xsl:for-each select="hotel">

  <xsl:for-each select="(1 to @sterne)">
    <img src="stern.gif" alt="*"/>
  </xsl:for-each><br />

</xsl:for-each>

</xsl:template>
```

XPath 2.0-Erweiterungen

- Inline-Schleifenkonstrukt for … in … return

```
<xsl:value-of select="for $i in (1 to 10) return $i * $i"/>
<xsl:value-of select="for $i in (1 to 5), $j in (5 to 10) return $i * $j"/>
```

- Beispiel Warenkorb: Summe aus Anzahl * Preis

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bestellungen>
  <auswahl>
    <produkt>A</produkt>
    <anzahl>2</anzahl>
    <preis>14.50</preis>
  </auswahl>
  <auswahl>
    <produkt>B</produkt>
    <anzahl>5</anzahl>
    <preis>9.20</preis>
  </auswahl>
  <auswahl>
    <produkt>C</produkt>
    <anzahl>1</anzahl>
    <preis>39.00</preis>
  </auswahl>
</bestellungen>
```

```
<xsl:value-of select="fn:sum(for $a in auswahl
return $a/anzahl * $a/preis)"/>
<!-- Ergebnis: 114 -->
```

1 vs. 20 Codezeilen
bei rekursiver Lösung
unter XSLT/XPath 1.0!

XPath 2.0-Erweiterungen

- Inline-Abfragetechnik if … then … else

```
<xsl:variable name="test" select="99"/>
<xsl:value-of select="if($test lt 100) then 'günstig' else 'teuer'"/>
```

  - reduziert den Einsatz von xsl:if- bzw. xsl:choose/xsl:when/xsl:otherwise-Blöcken
  - nützlich in Attribute Value Templates
    Beispiel alternierende CSS-Klassennamen (zelle_g bzw. zelle_u)

```
<xsl:for-each select="1 to 10">
  <td class="zelle_{if(position() mod 2 = 0) then 'g' else 'u'}">...</td>
</xsl:for-each>
```

- Kommentare in XPath-Ausdrücken (: … :)

```
<xsl:value-of select="1 to 5 (: Sequenz 1 bis 5 :)"/>
```

XPath 2.0-Erweiterungen

- Neue Knotentests: some/every … in … satisfies

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<daten>
  <eintrag>
    <x>1</x><y>10</y>
  </eintrag>
  <eintrag>
    <x>2</x><y>20</y>
  </eintrag>
  <eintrag>
    <x>3</x><y>30</y>
  </eintrag>
  <eintrag>
    <x>4</x><y>40</y>
  </eintrag>
  <eintrag>
    <x>5</x><y>50</y>
  </eintrag>
</daten>
```

```xml
<!-- Ausgabe von x-Inhalten,
     sofern irgendein x-Wert kleiner 10 ist -->
<xsl:if test="some $x in eintrag/x satisfies (fn:number($x) lt 10)">
 <xsl:for-each select="eintrag/x">
   <xsl:value-of select="."/><br />
 </xsl:for-each>
</xsl:if>

<!-- Ausgabe von y-Inhalten,
     sofern jeder y-Wert größer oder gleich 10 ist -->
<xsl:if test="every $y in eintrag/y satisfies (fn:number($y) ge 10)">
 <xsl:for-each select="eintrag/y">
   <xsl:value-of select="."/><br />
 </xsl:for-each>
</xsl:if>
```

XPath 2.0-Erweiterungen

- Mengenoperationen für Sequenzen (Knoten bzw. Werte)
  - $seq1 except $seq2 (Differenz = in 1, aber nicht in 2)
  - $seq1 intersect $seq2 (Durchschnitt = in 1 und 2)
  - $seq1 union $seq2 (Vereinigung = in einem von beiden)

```
<xsl:variable name="seq1" select="(1,3,5,7)"/>
<xsl:variable name="seq2" select="(2,3,5,8)"/>

<xsl:value-of select="$seq1 except $seq2" separator=","/>
<!-- 1,7 -->

<xsl:value-of select="$seq1 intersect $seq2" separator=","/>
<!-- 3,5 -->

<xsl:value-of select="$seq1 union $seq2" separator=","/>
<!-- 1,2,3,5,7,8 -->
```

XPath 2.0-Erweiterungen

- Neue Aggregat-Funktionen

fn:avg()	Mittelwert
fn:min()	Minimum
fn:max()	Maximum

ergänzen fn:sum() und fn:count() aus XPath 1.0

```
<xsl:variable name="seq" select="(10,20,30,40,50)"/>

<xsl:value-of select="fn:min($seq)"/><!--  10 -->
<xsl:value-of select="fn:max($seq)"/><!--  50 -->
<xsl:value-of select="fn:avg($seq)"/><!--  30 -->
<xsl:value-of select="fn:sum($seq)"/><!-- 150 -->
<xsl:value-of select="fn:count($seq)"/><!-- 5 -->
```

XPath 2.0-Erweiterungen

- Neue Zeichenketten-Funktionen

fn:ends-with(string,suchstring) in XPath 1.0 bereits fn:starts-with(...)

fn:string-join(string-sequence,separator) Zeichenketten zusammensetzen

fn:lower-case(string) Umwandlung in Kleinschreibung

fn:upper-case(string) Umwandlung in Großschreibung

```
<!-- XPath 2.0 -->
<xsl:value-of select="fn:lower-case($text)"/>
<xsl:value-of select="fn:upper-case($text)"/>

<!-- bisherige Ansätze auf Zeichenebene -->
<xsl:value-of select="translate($text,'ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ',
 'abcdefghijklmnopqrstuvwxyzäöü')"/>
<xsl:value-of select="translate($text,'abcdefghijklmnopqrstuvwxyzäöü',
 'ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ')"/>
```

XPath 2.0-Erweiterungen

- Zugriff auf Datum und Uhrzeit (XPath-Fkt.)
  fn:current-date() / fn:current-time() / fn:current-dateTime()

- Formatierung von Datum und Uhrzeit (XSLT-Fkt.)
  format-date() / format-time() / format-dateTime()

```
<xsl:variable name="datum" select="fn:current-date()"/>
<xsl:value-of select="format-date($datum,'[D01].[M01].[Y0001]')"/>
```

Bedeutung der Spezifikatoren (+ weitere Optionen für Schreibweisen):

Y	year	M	month in year
D	day in month	d	day in year
F	day of week	W	week in year
w	week in month	H	hour in day (24 hours)
h	hour in half-day (12 hours)	P	am/pm marker
m	minute in hour	s	second in minute
f	fractional seconds	Z	timezone

XPath 2.0-Erweiterungen

- Operationen mit Datums-/Zeitwerten

```
<xsl:variable name="datum_heute" select="fn:current-date()" as="xs:date"/>
<xsl:variable name="datum_ziel" select="xs:date('2007-11-07')" as="xs:date"/>

<xsl:value-of select="if($datum_ziel eq $datum_heute) then 'Zieldatum erreicht'
              else if($datum_ziel lt $datum_heute) then 'Zieldatum überschritten'
              else if($datum_ziel gt $datum_heute) then 'Zieldatum noch nicht erreicht'
              else()"/>
```

```
<xsl:variable name="datum_heute" select="fn:current-date()"/>
<xsl:variable name="weihnachten" select="xs:date(fn:concat(fn:year-from-date(
$datum_heute),'-12-24'))"/>
<xsl:value-of select="fn:days-from-duration($weihnachten - $datum_heute)"/><xsl:text>
Tage bis Weihnachten</xsl:text>
```

```
<xsl:text>Datum in 100 Tagen: </xsl:text>
<xsl:value-of select="format-date($datum_heute + 100 * xs:dayTimeDuration(
'P1D'),'[D01].[M01].[Y0001]')"/>

<xsl:text>Datum in 1 Jahr und 5 Monaten: </xsl:text>
<xsl:value-of select="format-date($datum_heute + xs:yearMonthDuration(
'P1Y5M'),'[D01].[M01].[Y0001]')"/>
```

Zeit-Perioden:
P1D = 1 Day
P1Y5M =
1 Year + 5 Months

XPath 2.0-Erweiterungen

- Verwendung von regulären Ausdrücken

fn:match(string, regex, flags?) — Suchen

fn:replace(string, regex, replacement, flags?) — Ersetzen

fn:tokenize(string, regex, flags?) — Trennen

```
<xsl:variable name="str" select="'D 06217'"/>

<xsl:value-of select="fn:matches($str,'(D )?\d{5}')"/>
<!-- true -->

<xsl:value-of select="fn:replace($str,'\d{5}','xxxxx')"/>
<!-- D xxxxx -->

<xsl:value-of select="fn:tokenize($str,' ')[2]"/>
<!-- 06217 -->
```

optionale Flags (z. B. mix)**:**
i = regex ist case-insensitive | **m** = multiline mode | **s** = **.** passt auf alle Zeichen, inkl. Newline (
) | **x** = alle Whitespace-Zeichen vor Anwendung entfernen

XPath 2.0-Erweiterungen

- Weitere nützliche Funktionen

fn:doc() und fn:doc-available() — externer Dokumentzugriff

```
<xsl:variable name="xmldoc" select="if(fn:doc-available('test.xml'))
then fn:doc('test.xml') else 'Dokument existiert nicht!'"/>
```

fn:document-uri() und fn:escape-html-uri() — URI-Referenzen

```
<xsl:value-of select="fn:document-uri($xmldoc)"/>
<xsl:value-of select="fn:escape-html-uri('http://www.example.net/müller/')"/>
                                    <!-- http://www.example.net/m%C3%BCller/ -->
```

fn:string-to-codepoints() / fn:codepoints-to-string() — Unicode

```
<xsl:value-of select="fn:string-to-codepoints('123')"/><!-- 49 50 51 -->
<xsl:value-of select="fn:codepoints-to-string((97,98,99))"/><!-- abc -->
<xsl:value-of select="fn:codepoints-to-string(8211)"/><!-- &#8211; -->
```

... und mehr (siehe Kurzreferenzen von P. Walmsley und D. Pawson)

XSLT 2.0-Erweiterungen

XSLT 2.0-Erweiterungen

- Verarbeitung von Textdateien

  XSLT-Funktion unparsed-text(filename,encoding)

```
<xsl:variable name="text" select="unparsed-text('buecher.txt','ISO-8859-1')"/>

<buecher>
  <xsl:for-each select="fn:tokenize($text,'(\r)|(\n)|(\r\n)')">
    <xsl:if test="fn:string-length(.) gt 0">
      <buch>
        <autor><xsl:value-of select="fn:substring-before(.,',')"/></autor>
        <titel><xsl:value-of select="fn:substring-after(.,',')"/></titel>
      </buch>
    </xsl:if>
  </xsl:for-each>
</buecher>
```

buecher.txt

```
Charles Bukowski,Barfly
George Orwell,1984
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<buecher>
   <buch>
      <autor>Charles Bukowski</autor>
      <titel>Barfly</titel>
   </buch>
   <buch>
      <autor>George Orwell</autor>
      <titel>1984</titel>
   </buch>
</buecher>
```

XSLT 2.0-Erweiterungen

- Benutzerdefinierte Funktionen: xsl:function
  - Element(e) unterhalb von xsl:stylesheet platzieren
  - Argumente als Parameter (xsl:param), optionale Argumente nur über mehrfache Funktionsdeklaration möglich
  - Funktionsname mit eigenem Namensraum verknüpft, z. B. ns:fktname
  - Rückgabe von atomaren Werten oder Sequenzen
  - Anwendung innerhalb von XPath-Ausdrücken
  - Attribut override (yes|no) ermöglicht Überschreiben Prozessor-eigener Funktionen mit identischem Namen (z. B. saxon:sort)

```
<!-- Fakultät -->
<xsl:function name="tm:fact" as="xs:double">

  <xsl:param name="n" as="xs:integer"/>
  <xsl:value-of select="if($n eq 0 or $n eq 1) then 1 else $n * tm:fact($n - 1)"/>

</xsl:function>
```

```
<!-- Aufruf -->
<xsl:text>5! = </xsl:text><xsl:value-of select="tm:fact(5)"/>
<!-- 120 -->
```

XSLT 2.0-Erweiterungen

- Benutzerdefinierte Funktionen: xsl:function

  Beispiel Kreisdiagramm aus XML-Daten über Winkelfunktionen

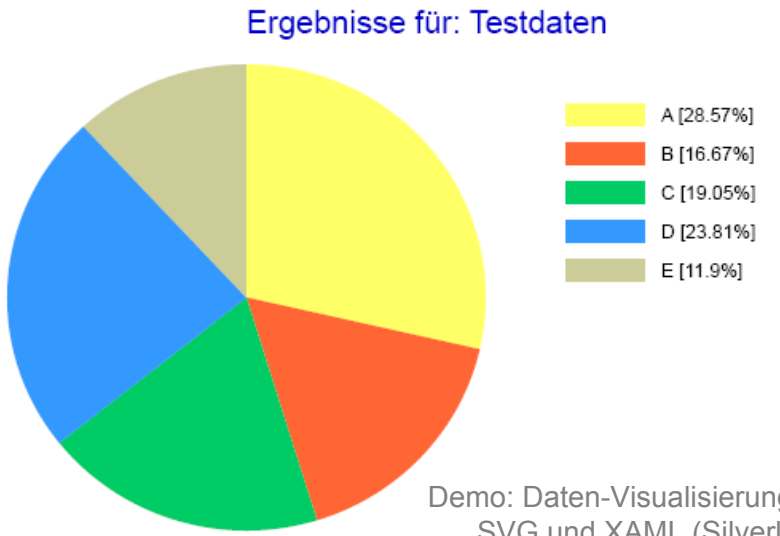
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<daten info="Testdaten">
  <satz>
    <wert>120</wert>
    <text>A</text>
    <farbe>#FF6</farbe>
  </satz>
  <satz>
    <wert>70</wert>
    <text>B</text>
    <farbe>#F63</farbe>
  </satz>
  <satz>
    <wert>80</wert>
    <text>C</text>
    <farbe>#0C6</farbe>
  </satz>
  <satz>
    <wert>100</wert>
    <text>D</text>
    <farbe>#39F</farbe>
  </satz>
  <satz>
    <wert>50</wert>
    <text>E</text>
    <farbe>#CC9</farbe>
  </satz>
</daten>
```

Sinus

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

```
<xsl:function name="tm:sin" as="xs:double">...</xsl:function>
<!-- analog Cosinus-Funktion als tm:cos ... -->
```

tm_mathlib.xsl



Demo: Daten-Visualisierung als SVG und XAML (Silverlight)

XSLT 2.0-Erweiterungen

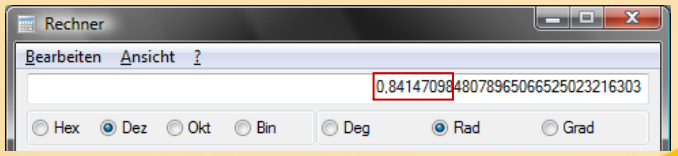
- Benutzerdefinierte Funktionen: xsl:function

  Umsetzung der Sinus-Funktion über eine Reihenentwicklung

```
<!-- Sinus-Funktion
     globale Variablen $pi und $max | separate Funktionen tm:fact() und tm:pow() -->

<xsl:function name="tm:sin" as="xs:double">

  <xsl:param name="arg" as="xs:double"/>

  <xsl:variable name="x" as="xs:double" select="if($arg ge 2 * $pi) then
   $arg - fn:floor($arg div (2 * $pi)) * 2 * $pi else(if($arg le -2 * $pi) then
   $arg + fn:floor($arg div (-2 * $pi)) * 2 * $pi else $arg)"/>

  <xsl:variable name="sum_seq" as="item()*" select="for $n in (0 to $max) return
   fn:number(tm:pow(-1, $n) * tm:pow($x, 2 * $n + 1) div tm:fact(2 * $n + 1))"/>

  <xsl:value-of select="fn:round-half-to-even(fn:sum($sum_seq), 8)"/>

</xsl:function>
```

neue Rundungsfunktion

```
<xsl:value-of select="tm:sin(1)"/>
<!-- 0.84147098 mit $max = 15 -->
```



XSLT 2.0-Erweiterungen

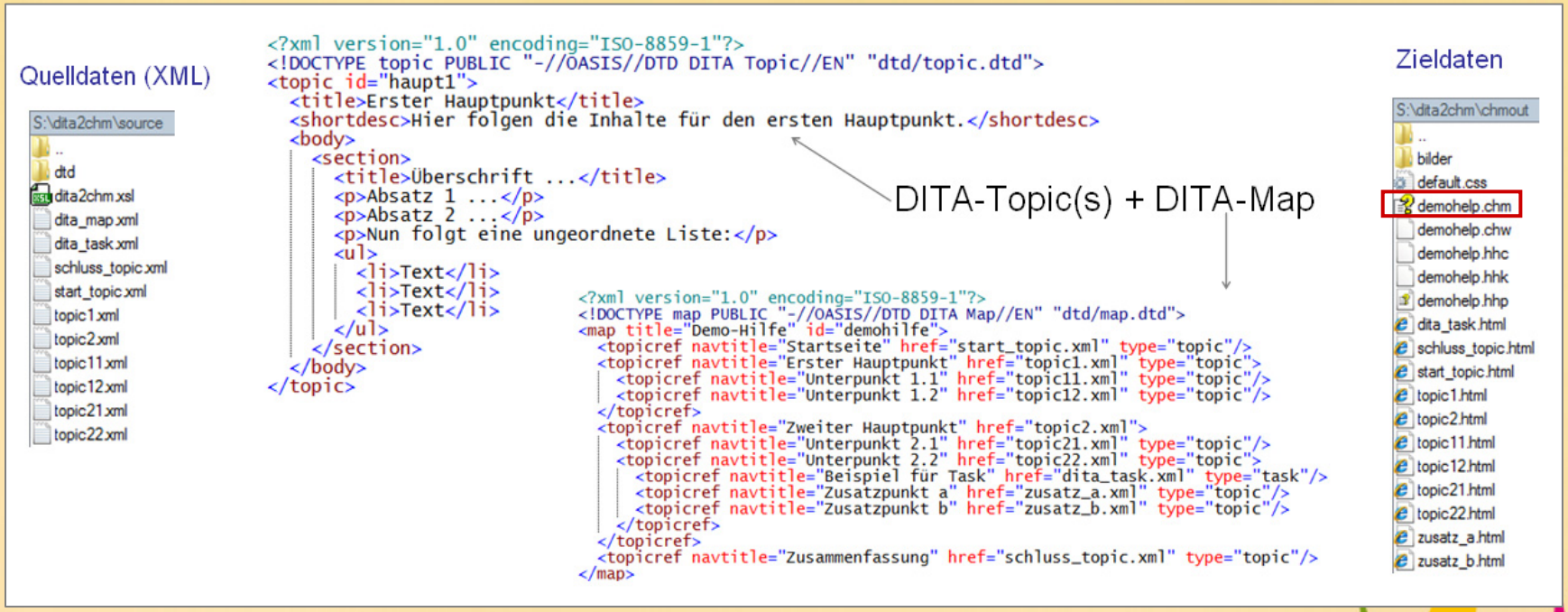
- Mehrfachausgabe: xsl:result-document
  - ermöglicht die Ausgabe beliebig vieler Ergebnisdokumente innerhalb einer Transformation
  - Anlage mehrerer xsl:output-Elemente unterhalb von xsl:stylesheet unter Verwendung eines Namens (name) für die spätere Referenzierung
  - xsl:result-document verweist über das format-Attribut auf die deklarierten Namen und erhält den Dateinamen für die Ausgabe als href-Attribut

```
<xsl:output method="html" name="web" ... />
<xsl:output method="text" name="csv" ... />
<xsl:output method="xml"  name="svg" ... />

<xsl:template match="...">

  <!-- HTML-Ausgabe -->
  <xsl:result-document href="ergebnis.html" format="web"><html>...</html></xsl:result-document>

  <!-- Text-Ausgabe -->
  <xsl:result-document href="ergebnis.txt" format="csv"><xsl:text>...</xsl:text></xsl:result-document>

  <!-- SVG-Ausgabe -->
  <xsl:result-document href="ergebnis.svg" format="svg"><svg>...</svg></xsl:result-document>

</xsl:template>
```

XSLT 2.0-Erweiterungen

- Mehrfachausgabe: xsl:result-document

  Beispiel CHM-Online-Hilfe auf DITA-Basis (Topics und Map)



Generierung von hhp-, hhc- und hhk-Dateien für den Help-Compiler (hhc.exe) sowie je einer (X)HTML-Datei pro Topic über die DITA-Map.

Details im Entwickler Magazin 6.2007 → Demo von dita2chm.xsl

XSLT 2.0-Erweiterungen

- Zeichenersetzung: xsl:character-map
  - Ausgabe spezieller Zeichenfolgen (xsl:output-character) ohne umständliche Maskierungen mittels disable-output-escaping="yes"
  - Belegung von Zeichencodes der privaten Unicode-Areas (u. a. E000-F8FF) möglich und zusätzliche Deklaration von Entitys im Stylesheet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY phpstart "&#xE001;">
<!ENTITY phpende  "&#xE002;">
]>
<xsl:stylesheet version="2.0" ...>

<xsl:output method="text" encoding="ISO-8859-1" indent="yes"
  use-character-maps="map1 map2"/>

<xsl:character-map name="map1">
  <xsl:output-character character="&#xE001;" string="&lt;?php "/>
  <xsl:output-character character="&#xE002;" string=" ?&gt;"/>
</xsl:character-map>

<xsl:character-map name="map2">
  <xsl:output-character character=" " string="&amp;nbsp;"/>
  <xsl:output-character character="&#8211;" string="--"/>
</xsl:character-map>
...
</xsl:stylesheet>
```

Anwendung als
&phpstart;
bzw.
&phpende;

XSLT 2.0-Erweiterungen

- Zeichendaten-Analyse: xsl:analyze-string
  - Kindelemente xsl:matching-substring und xsl:non-matching-substring
  - Verwendung regulärer Ausdrücke (Attribute regex und optional flags)
  - Abfragen von Gruppen mit der Funktion regex-group()

```
<xsl:variable name="test" select="'D 06217'"/><!-- PLZ -->

<xsl:analyze-string select="$test" regex="(D )?(\d{{5}})">

 <xsl:matching-substring>
   <xsl:text>PLZ ok</xsl:text><br />
   Landeskennung: <xsl:value-of select="regex-group(1)"/><br />
   Zahlencode: <xsl:value-of select="regex-group(2)"/>
 </xsl:matching-substring>

 <xsl:non-matching-substring>
   <xsl:text>fehlerhafte PLZ</xsl:text>
 </xsl:non-matching-substring>

</xsl:analyze-string>
```

{…} steht für AVT, deshalb im Ausdruck maskiert!

PLZ ok

Landeskennung: D

Zahlencode: 06217

XSLT 2.0-Erweiterungen

- Gruppierung: xsl:for-each-group
  - einfacher als die übliche Methode nach S. Muench (Basis xsl:key)
  - Zugriff auf Gruppen mit Funktion current-group()
  - Zugriff auf Schlüssel mit Funktion current-grouping-key()
  - Attribute zum Gruppieren nach
    - Inhalten/Werten: group-by
    - benachbarten Knoten: group-adjacent
    - Startknoten: group-starting-with
    - Endknoten: group-ending-with

XSLT 2.0-Erweiterungen

- Gruppierung: xsl:for-each-group → Beispiel

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<technologien>
  <technologie>
    <name>CSS</name><herausgeber>W3C</herausgeber>
  </technologie>
  <technologie>
    <name>HTML</name><herausgeber>W3C</herausgeber>
  </technologie>
  <technologie>
    <name>DITA</name><herausgeber>OASIS</herausgeber>
  </technologie>
  <technologie>
    <name>DocBook</name><herausgeber>OASIS</herausgeber>
  </technologie>
  <technologie>
    <name>JavaScript</name><herausgeber>ECMA</herausgeber>
  </technologie>
  <technologie>
    <name>ODF</name><herausgeber>ISO</herausgeber>
  </technologie>
  <technologie>
    <name>XPS</name><herausgeber>ECMA</herausgeber>
  </technologie>
  <technologie>
    <name>XSLT</name><herausgeber>W3C</herausgeber>
  </technologie>
</technologien>
```

```xml
<xsl:for-each-group select="technologie"
  group-by="herausgeber">

  <xsl:sort select="herausgeber" data-type="text"
    order="ascending"/>

    <h1><xsl:value-of select="herausgeber"/></h1>
    <ul>
    <xsl:for-each select="current-group()">
      <li><xsl:value-of select="name"/></li>
    </xsl:for-each>
    </ul>

</xsl:for-each-group>
```

ECMA
- JavaScript
- XPS

ISO
- ODF

OASIS
- DITA
- DocBook

W3C
- CSS
- HTML
- XSLT

XSLT 2.0-Erweiterungen

- Schema-Validierung: xsl:import-schema
  - Import von Schema-Dokumenten (XSD) zur Validierung der erzeugten Ausgaben oder zur Prüfung von eigenen Datentypen
  - Element xsl:import-schema unterhalb von xsl:stylesheet platzieren und Attribut schema-location belegen
  - Validierung mittels validation-Attribut für Elemente wie xsl:document, xsl:result-document oder xsl:copy bzw. xsl:copy-of vorgesehen

```
<xsl:import-schema namespace="http://www.w3.org/1999/xhtml"
  schema-location="http://www.w3.org/2002/08/xhtml/xhtml1-strict.xsd"/>

<xsl:template match="technologien">
<xsl:document validation="strict"><!-- strict | lax | preserve | strip -->
  <html xmlns="http://www.w3.org/1999/xhtml" lang="de" xml:lang="de">
  <head><title>Vortragsmaterial zu XSLT/XPath 2.0</title></head>
  <body><p>Einige Technologien:</p>
  <ul>
    <xsl:for-each select="technologie">
      <li><xsl:value-of select="name"/></li>
    </xsl:for-each>
  </ul>
</body></html>
</xsl:document>
</xsl:template>
```

XSLT 2.0-Erweiterungen

- Erzeugung von Teildokumenten: xsl:document
  - Aufbau von Teilstrukturen für die spätere Ausgabe oder den temporären Zugriff auf Inhalte
  - optionale Validierung von einzelnen Ergebnisbäumen vor der Ausgabe

```
<xsl:variable name="dok_baum" as="document-node()">

  <xsl:document>
    <test>
      <abc>1</abc>
      <abc>2</abc>
      <abc>3</abc>
    </test>
  </xsl:document>

</xsl:variable>

<!-- temp. Zugriff auf Teilstrukturen: -->
<xsl:value-of select="$dok_baum/test/abc[1]/text()"/><!-- 1 -->

<!-- Ausgabe der erzeugten Teilstruktur: -->
<xsl:copy-of select="$dok_baum"/>
```

XSLT 2.0-Erweiterungen

- Sortierung von Sequenzen: xsl:perform-sort
  - spart xsl:for-each-Konstrukte, nützlich vor allem innerhalb von Funktionen
  - Kindelement xsl:sort (Anwendung wie aus XSLT 1.0 bekannt)

```
<!-- Nachbildung der Saxon-Erweiterungsfunktion saxon:sort()
     override="no" bedeutet: verwende das Original, sofern verfügbar -->
<xsl:function name="saxon:sort" as="item()*" override="no">

  <xsl:param name="seq" as="item()*"/>
  <xsl:param name="ord" as="xs:string"/>
  <xsl:param name="typ" as="xs:string"/>

  <xsl:perform-sort select="$seq">
    <xsl:sort select="." order="{$ord}" data-type="{$typ}"/>
  </xsl:perform-sort>

</xsl:function>
```

```
<xsl:value-of select="saxon:sort((8,5,7,3,4,2,9,1,6),'descending','number')"/>
<!-- 9 8 7 6 5 4 3 2 1 -->
<xsl:value-of select="saxon:sort(('Welt!','Hallo'),'ascending','text')"/>
<!-- Hallo Welt! -->
```

XSLT 2.0-Erweiterungen

- Neue Formatoptionen für xsl:number
  - Attribut format="…" kann mit den Kürzeln w, W, Ww Zahlworte ausgeben (one, two, … / ONE, TWO, … / One, Two, …)
  - mit dem Attribut ordinal="yes" ergibt sich die Form first, second, …
  - zusätzliche Angabe von lang="de" für Sprachanpassung (eins / erste)
  - Angabe format="1" und ordinal="yes" führt zu 1st, 2nd, 3rd, …

```
<!-- zum Beispieldokument technologien.xml -->
<xsl:for-each select="technologie">

  <xsl:number format="Ww: " ordinal="yes" lang="de"/>
  <xsl:value-of select="name"/><br />

</xsl:for-each>
```

Erste: CSS
Zweite: HTML
Dritte: DITA
Vierte: DocBook
Fünfte: JavaScript
Sechste: ODF
Siebte: XPS
Achte: XSLT

XSLT 2.0-Erweiterungen

- Sonstige Erweiterungen
  - xsl:namespace

```
<svg version="1.1">
  <xsl:namespace name="svg">http://www.w3.org/2000/svg</xsl:namespace>
  <xsl:namespace name="xlink">http://www.w3.org/1999/xlink</xsl:namespace>
  <a xlink:href="http://example.net/"><circle cx="50" cy="50" r="20" fill="red"/></a>
</svg>
```

  - Vorgabe-Namensraum

```
<!-- bezogen auf bestimmten NS im Eingabe-XML-Dokument, spart Prefixe im XSL -->
<xsl:stylesheet version="2.0" ... xpath-default-namespace="...">
...
</xsl:stylesheet>
```

  - Namensräume (nicht) mit kopieren [xsl:copy / xsl:copy-of]

```
<xsl:template match="node() | @*">
  <xsl:copy copy-namespaces="no">
    <xsl:apply-templates select="node() | @*"/>
  </xsl:copy>
</xsl:template>
```

XSLT 2.0-Erweiterungen

- Sonstige Erweiterungen
  - neue Attribute für xsl:param → required bzw. tunnel (zur impliziten Übergabe)

```
<xsl:param name="x" required="yes"/><!-- Standardwert no -->
...
<xsl:param name="y" tunnel="yes"/><!-- Standardwert no -->
<!-- tunnel-Attribut auch für xsl:with-param anwendbar -->
```

  - Ergebnis Unendlich (INF / -INF) und (+)vorzeichenbehaftete Werte

```
<xsl:variable name="x" select="1.234567890E123" as="xs:double"/>
<xsl:value-of select="$x * $x"/><!-- 1.52415787501905E246 -->
<xsl:value-of select="$x * $x * $x"/><!-- INF -->
```

```
<xsl:variable name="x" select="123" as="xs:integer"/>
<xsl:value-of select="-$x + $x"/><!-- 0 -->
<xsl:value-of select="+$x - $x"/><!-- 0, unter XPath 1.0 ungültiger Ausdruck (+$x) -->
```

  - xsl:next-match
    gezielter Aufruf des am nächsten relevanten Templates

XSLT 2.0-Erweiterungen

- Sonstige Erweiterungen
  - Rückwärts-Kompatibilität und Feature-Tests
    - Attribut version für alle xsl-Elemente verfügbar (bei xsl:output Bedeutung für die Ausgabe)
    - durch Angabe von version="1.0" kann 1.0-Verhalten gefordert werden
    - gezielte Ablaufsteuerung mit dem Attribut use-when
    - Test mit function-available('fn:tokenize')
    - Test mit element-available('xsl:result-document')
    - Test mit system-property('xsl:is-schema-aware' = 'yes')
    - Test mit fn:number(system-property('xsl:version')) >= 2.0

```
<xsl:value-of select="technologie" version="2.0"/>
<!-- CSS HTML DITA DocBook JavaScript ODF XPS XSLT -->
<xsl:value-of select="technologie" version="1.0"/><!-- CSS -->
```

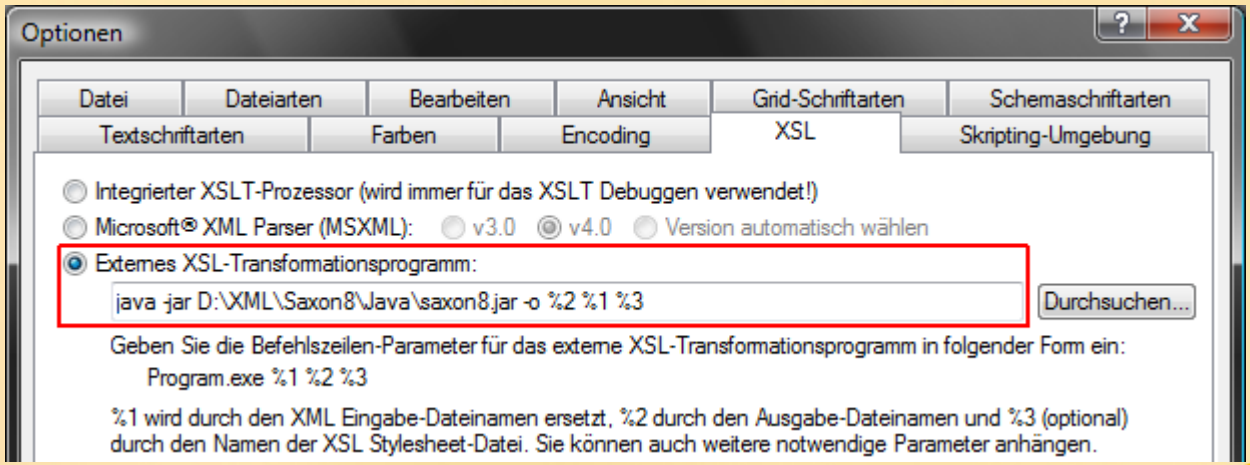
```
<xsl:template match="technologien"
  use-when="fn:number(system-property('xsl:version')) >= 2.0">...</xsl:template>
```

  - Debugging mit Funktion fn:trace()

```
<xsl:variable name="xyz" select="1.95583"/>
<xsl:value-of select="fn:trace($xyz,'Wert der Variable $xyz: ')"/>
```

XSLT 2.0-Erweiterungen

- Aufruf von AltovaXML und Saxon
  - pfad/zu/AltovaXML.exe -xslt2 name.xsl -in name.xml -out name.html
  - java -jar pfad/zu/saxon8.jar -o name.html name.xml name.xsl
    bzw.
    pfad/zu/Transform.exe -o name.html name.xml name.xsl
  - Einbindung externer Prozessoren in XMLSpy:



XSLT 2.0 und XPath 2.0

- Zusammenfassung
  - XSLT 2.0 und XPath 2.0 bieten gegenüber den acht Jahre alten 1.0-Versionen umfangreiche Erweiterungen, Verbesserungen und Vereinfachungen
  - Transformationsvorlagen lassen sich kompakter formulieren und sollten insgesamt effizienter ausgeführt werden
  - Entwickler profitieren nach kurzer Einarbeitung von zahlreichen neuen XSLT-Elementen und XPath-Funktionen
  - Unterstützung durch die Prozessoren wird sich in der nächsten Zeit verbessern
  - Künftige .NET-Frameworks sollen Implementierungen bieten, entsprechende Pläne für PHP 6 sind noch nicht bekannt
  - Web-Browser arbeiten noch auf 1.0-Niveau

XSLT 2.0 und XPath 2.0

- Referenzen
  - W3C-Spezifikationen (Links siehe Folie 14)
  - Walmsley, P.: FunctX; http://www.xqueryfunctions.com/
  - Pawson, D.: Quick-Ref.; http://www.dpawson.co.uk/xsl/rev2/xslt2.pdf
  - Software Saxon: http://saxon.sourceforge.net/
  - Software AltovaXML: http://www.altova.com/altovaxml.html
  - Winkelfunktionen: http://de.wikipedia.org/wiki/Sinus_und_Kosinus
  - DITA2CHM: http://datenverdrahten.de/xslt2/dita2chm.html
  - Beispiele: http://www.iks.hs-merseburg.de/~meinike/vortraege.php
- Kontakt
  - E-Mail: thomas.meinike@hs-merseburg.de
  - WWW: http://www.iks.hs-merseburg.de/~meinike/